Context switch

1. Hårdvara:
   a) Avbrottsystemet slås av.
   b) CPU–status, PC, SP sparas
   c) Supervisor mode slås på
2. Operativsystem:
   a) Register, processtatus etc sparas i PCB
   b) Vad hände? Systemanrop eller fel?
   c) Åtgärda: snabba saker direkt, annars väntelista
   d) Välj ny process från readykön (schemaläggning)
3. Dispatch:
   a) Återställ register, CPU–status, SP, avbrottsystem etc
   b) och PC (nu kör den nya processen)

Processkommunikation

Implementationsfrågor:

- Hur skapas en länk?
- Hur många processer kan dela en länk?
- Hur många länkar kan finnas mellan 2 proc?
- Hur många meddelanden kan finnas i länken?
- Hur stora kan meddelanden vara?
- Är länkar enkel– eller dubbelriktade?

Producent/konsument

```
<producera>                 while (full == 0)
while (full == N)               noop;
    noop;                   nextc = buf[out];
buf[in]=nextp;              out = (out+1)%N;
in = (in+1)%N;              full = full–1;
full = full+1;              <konsumera>
```

Producent/konsument

```
<producera>                 while (full == 0)
while (full == N)               noop;
    noop;                   nextc = buf[out];
buf[in]=nextp;              out = (out+1)%N;
in = (in+1)%N;              full = full–1;
full = full+1;              <konsumera>
```

Före: antag full=5

load reg <= full;	load reg <= full;
increment reg;	decrement reg;
store full <= reg;	store full <= reg;

Efter: full $\in \{4,5,6\}$

Critical section problem

- N processer delar data.
- varje process har en kritisk sektion (CS) där delade data uppdateras.
- Problemet: se till att när en process är i sin kritiska sektion får ingen annan vara i sin kritiska sektion.

Critical section problem

Krav på lösning:

1. **Mutual exclusion:** Om process kör i sin CS så kan ingen annan proc köra i sin CS.
2. **Progress** (framåtskridande): Om ingen proc är i sin CS och någon proc vill in i sin CS så kan valet av proc inte skjutas upp för alltid.
3. **Bounded waiting** (begränsad väntan): Det måste finnas en gräns för hur många ggr andra proc får gå in i sina CS efter att en proc har börjat vilja gå in och innan den får komma in.

Critical section problem

Antaganden för en lösning:

- ingen process står still
- inget antagande om relativ hastighet hos proc.
- load, store, och test–instruktioner är atomära

CS problem

Processtruktur:

```
repeat
    entry section          – be att få komma in
    critical section
    exit section           – avsluta
    remainder section      – icke–kritisk kod
until false
```

Test–and–set

```
ATOMIC boolean test_and_set(loc)
{
    boolean x = loc;
    loc = true;
    return x;
}
```

CS lösning med test_and_set

```
lock = false;
repeat
    while test_and_set(lock)
        ;
    critical section
    lock = false;
    remainder section
until false
```

Swap

```
ATOMIC swap(a,b)
{
    x = a;
    a = b;
    b = x;
}
```

CS lösning med swap

```
lock = false;
repeat
    key = true;
    repeat swap(lock,key)
        until (key == false);
    critical section
    lock = false;
    remainder section
until false
```

Semaforer

```
// s ≥ 0: hur många wait() utan att blockeras
// s < 0: |s| är antalet blockerade processer

ATOMIC wait(s) {
    s = s–1;
    if (s < 0) then block(s);          – suspendera P
}
ATOMIC signal(s) {
    s = s+1;
    if (s <= 0) then wakeup(s);        – väck en proc
}
```

CS lösning med semafor

```
Semafor mutex = 1;
repeat
    wait(mutex);
    critical section
    signal(mutex);
    remainder section
until false
```

Synkronisering med semaforer

```
// anta S1 ska göras före
  S2
Semafor synch = 0;

P1: S1;                          P2: wait(synch);
    signal(synch);                   S2;
```

Problem fortfarande möjliga

Semafor S = 1, Q = 1;

```
P1: wait(S);          P2: wait(Q);
    wait(Q);              wait(S);
    ....                  ....
    signal(S);            signal(Q);
    signal(Q);            signal(S);
```